

SDEV 1201

Database Fundamentals

LESSON 7

Query Syntax

Queries are written with the SELECT statement. The simplified syntax for a select statement is:

```
SELECT [ALL | DISTINCT] column_list
[FROM table_name [, table_name2 [... , table_name]] -- source of data
      [INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN] -- other sources
[WHERE clause] -- filter
[GROUP BY clause] -- aggregate by
[HAVING clause] -- filter by aggregate
[ORDER BY clause] -- sorts results
```

If it looks complex don't worry. All the "clauses" listed don't need to be used when creating a query. However, what clauses you do use MUST be in the order shown above (i.e. HAVING MUST be after GROUP BY for example and not vice versa).

Simple SELECT

```
SELECT  
    FirstName  
, LastName  
, City  
FROM Student;
```

Like

The Like operator is used when you use wildcards in your search or perform pattern matching. Wildcards are characters that when used with the Like operator allow the wildcards to represent a character or multiple characters. The Like operator is case sensitive in PostgreSQL.

For example, if we wanted to return all the student's names whose names started with the letter 'D' we would use the % wildcard character. The % wildcard means any character and any number of characters.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName Like 'D%';
```

It is important to use the like operator when using wildcards. The % would be interpreted as a literal '%' if we used = instead of Like.

Like

The `_` (underscore) is a single character wildcard. This would be used when you are looking for any one character in your search criteria. If we wanted the students' names whose first name was 3 characters long and started with 'Ja' we could use this query.

```
Select FirstName || ' ' || LastName "Student Name"  
From Student  
Where FirstName Like 'Ja_';
```

If we had used a `%` instead of `_` we would also return students whose First Name was longer than 3 characters.

Aggregate Functions

Aggregate functions calculate a single value using the data from many records. If you have used function in Excel such as SUM, AVERAGE, COUNT, etc.... then you have used aggregate functions.

aggregate_function_name ([ALL | DISTINCT] expression)

- *AVG(ColumnName)* returns the average of numeric values. **NULL** is ignored.
- *SUM(ColumnName)* returns the sum of a column containing numeric values.
- *MIN(ColumnName)* and *MAX(ColumnName)* returns the minimum & maximum values from a column of numeric, date, or character values.
- *COUNT(*)* returns the number of records that match the **WHERE** criteria.
- *COUNT(ColumnName)* returns the number of non-null values in a given column for records that match the **WHERE** criteria.

String Functions

- `LENGTH(column | expression)`
- `LEFT(column | expression, length)`
- `RIGHT(column | expression, length)`
- `SUBSTRING(column | expression, start, length)`
- `REVERSE(column | expression)`
- `UPPER(column | expression)`
- `LOWER(column | expression)`
- `LTRIM(column | expression)`
- `RTRIM(column | expression)`

Date Functions

- `Current_Date` returns the current system date
- `Current_Time` returns the current system time
- `DATE_PART(xx, date1)` returns integer representation of the `xx` of `date1`

`xx` represents field for date portions (see next slide).

Date Function Examples

-- You can also use Fields to get the character representation of dates.

`Select To_Char (Current_Date, 'DAY');`

Returns the day of the week (i.e. MONDAY, TUESDAY, etc.). Note that the “Field” is case sensitive: “Day” would return “Monday”.

`Select To_Char (Current_Date, 'MONTH');`

Returns the month name. Note that the “Field” is case sensitive: “Month” would return “September”.

Group By

The **GROUP BY** clause is used with aggregate functions to provide subtotals. We could easily return the average Mark from the grade table. What if we wanted the average Mark for each course? For each student? For each course for each Student? To do this we would use GROUP BY.

```
SELECT StudentID, AVG(Mark) AS AverageMark  
FROM Registration  
GROUP BY StudentID
```

calculates the average mark per Student.

We will usually GROUP BY something unique.

Another way to think of the syntax GROUP BY is the 'for each'.

Having

HAVING is like the **WHERE** clause, except it applies its criteria after **GROUP BY**. For example:

```
SELECT StudentID, AVG(Mark) as AverageMark
FROM Registration
GROUP BY StudentID
WHERE AVG(Mark) > 80
```

Doesn't work. However, Replacing **WHERE** with **HAVING** having does.

```
SELECT StudentID, AVG(Mark) as AverageMark
FROM Registration
GROUP BY StudentID
HAVING AVG(Mark) > 80
```

Joins

For example, if I want to select some fields from the Staff table and other fields from the Position table, which are connected using the PositionID PK/FK

```
select FirstName || ' ' || LastName "Staff Name",  
PositionDescription "Position"  
from Staff  
inner join Position  
on Staff.PositionID = Position.PositionID
```

The INNER JOIN syntax tells the server that we are performing an inner join and identifies that PositionID is the field in the 2 tables that relate them together.

Selecting from 3 or more tables

To select from more than 2 tables the syntax would look like:

```
SELECT field1, field2,... FROM table1  
INNER JOIN table2  
ON table1.joinfield = table2.joinfield  
INNER JOIN table3  
ON table.joinfield = table.joinfield  
INNER JOIN table4  
ON table.joinfield = table.joinfield
```

Selecting from 3 or more tables

Select the Student Names, Payment Dates, and Payment Type Descriptions for all the students that have payments:

```
select Student.StudentId, FirstName || ' ' || LastName "Student Name",  
PaymentDate, PaymentTypeDescription  
from Student  
inner join Payment  
on Student.StudentID = Payment.StudentID  
inner join PaymentType  
on Payment.PaymentTypeID = PaymentType.PaymentTypeID
```

Solution

Inner Join With Aggregates Exercises found on the Brightspace site in the “Week 4” Section.

Today's Objective

Today we will be covering:

- ❑ Outer Joins
- ❑ Lab time

Theory and reference material for this presentation is based on the “Advanced Select Statement” document in the “Week 4 - Select Statements” section of Brightspace.

Outer Joins

Joins

```
Select Staff.FirstName, Staff.LastName, Position.PositionDescription  
From Staff  
Inner Join Position  
On Staff.PositionID = Position.PositionID;
```

An Outer Join is different from an Inner Join. An Inner Join only returns records from the parent table that have related records in the child table. Looking back at the results you will notice that one Position Description is not returned in the results. Position 7 (Assistant Dean) was not returned because there are no Staff with that position. With an Inner Join parent records that do not have a matching child record are not returned.

An Outer Join is different in that it returns all the parent records, including the ones that do not have a matching record in the related child table. To return ALL the Position Descriptions and the staff names that are in those positions you would use an Outer Join.

Joins

An Outer Join is different in that it returns all the parent records, including the ones that do not have a matching record in the related child table. To return ALL the Position Descriptions and the staff names that are in those positions you would use an Outer Join. We will discuss Left Outer Joins and Right Outer Joins. These both have the same purpose, and you choose which one to use based on the order you list your tables in your select statement.

```
Select Staff.FirstName, Staff.LastName, Position.PositionDescription  
From Position  
Left Outer Join Staff  
On Staff.PositionID = Position.PositionID ;
```

This example states to select ALL the records from the table that is on the LEFT side of the OUTER JOIN statement (Position) and the related records from Staff. The results will include the Assistant Dean position description and a NULL value for staff name.

Joins

```
Select Staff.FirstName, Staff.LastName, Position.PositionDescription  
From Staff  
Right Outer Join Position  
On Staff.PositionID = Position.PositionID;
```

This example states to select ALL the records from the table that is on the RIGHT side of the OUTER JOIN statement (Position) and the related records from Staff.

SDEV1201 Coding Standard

In labs and quizzes, when doing an outer join the syntax “outer join” must be used for clarity.

Difference between INNER and OUTER join

The difference between Inner Join and Outer Join:

```
Select Student.FirstName, Student.LastName, Payment.PaymentDate  
From Student  
Inner Join Payment  
On Student.StudentID = Payment.StudentID;
```

(returns 33 records)

```
Select Student.FirstName, Student.LastName, Payment.PaymentDate  
From Student  
Left Outer Join Payment  
On Student.StudentID = Payment.StudentID;
```

(returns 42 records)

Can you explain the difference?

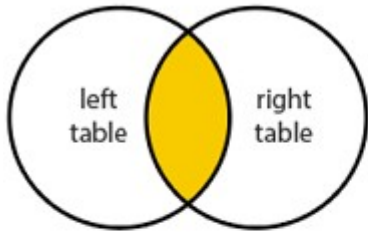
The difference is there are only 33 students that have made payments (only 33 parent records that have matching child records) so only 33 records returned.

An Inner Join only returns Parent records that have matching child records.

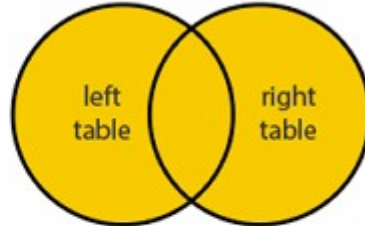
An Outer Join returns ALL the Parent records (students) even if they do not have any matching child records (payments). Therefore we now know that 9 students have no payment records. Since there were no payment dates for those students the PaymentDate field for those records contains the value of NULL.

Types of JOINS

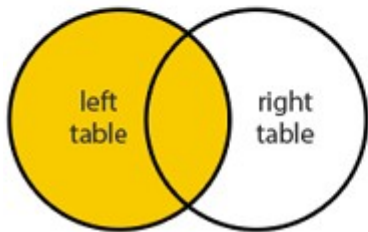
INNER JOIN



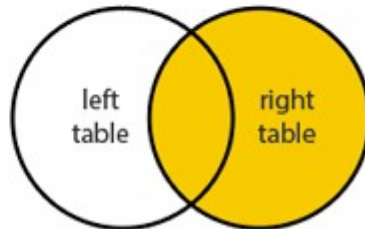
FULL JOIN



LEFT JOIN

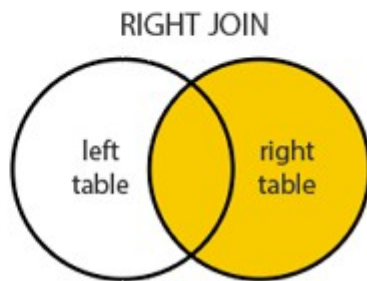
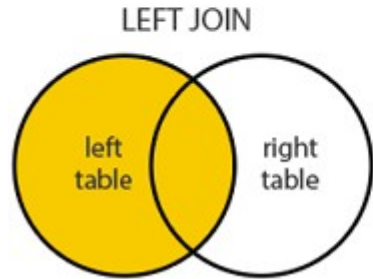
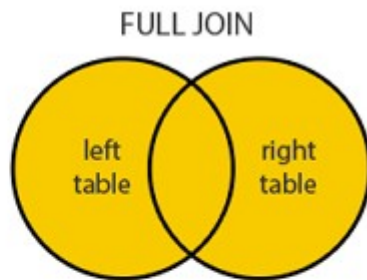
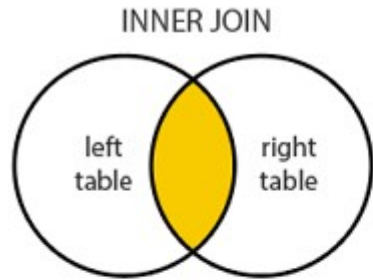


RIGHT JOIN



- **INNER JOIN** returns only records that exist in both tables
- **LEFT OUTER JOIN** returns all records in **table1**, regardless of whether they exist in **table2**
- **RIGHT OUTER JOIN** returns all records in **table2**, regardless of whether they exist in **table1**
- **FULL OUTER JOIN** returns all records that exist in either table (and is seldom used)

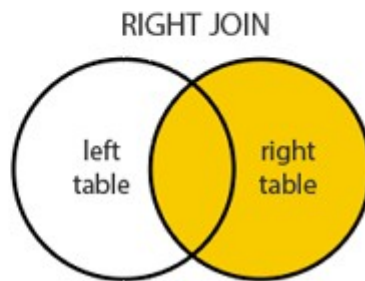
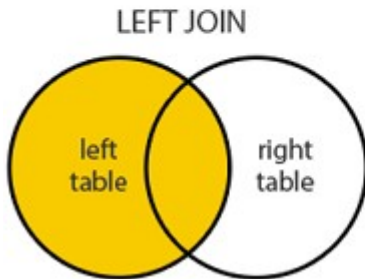
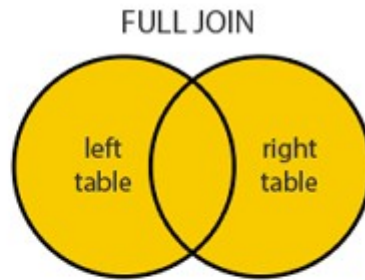
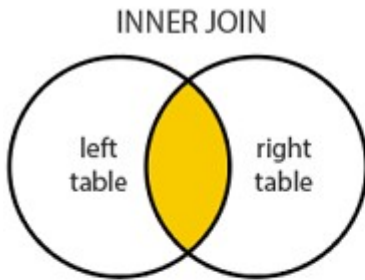
Types of JOINS



- **INNER OUTER JOIN** returns only records that exist in both tables

If we're joining a parent and child, this means we only see records for parents that have children.

Types of JOINS

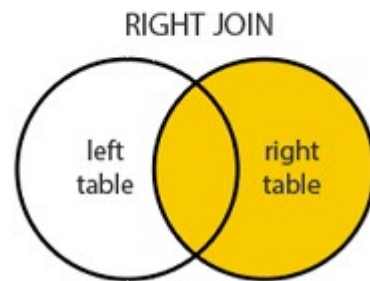
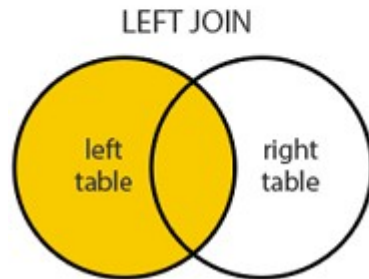
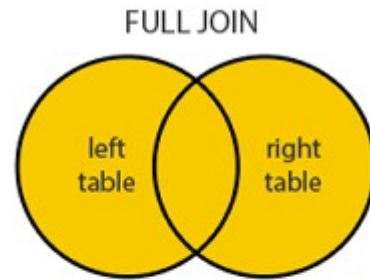
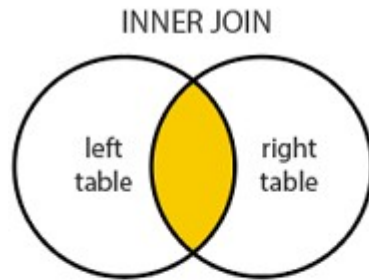


- **LEFT OUTER JOIN** returns all records in *table1*, regardless of whether they exist in *table2*

Parent **LEFT OUTER JOIN** *Child* returns parents regardless of whether they have child records.

Child **LEFT OUTER JOIN** *Parent* returns child records, so we should use **INNER JOIN** instead.

Types of JOINS

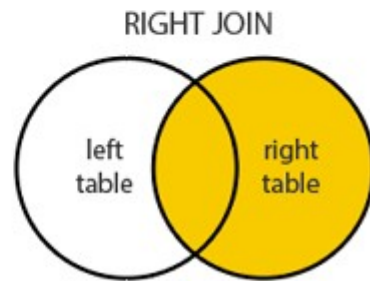
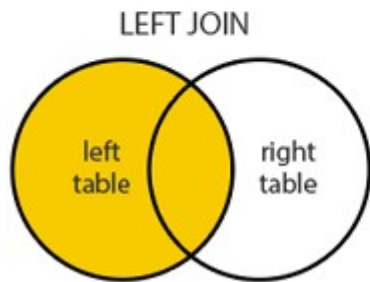
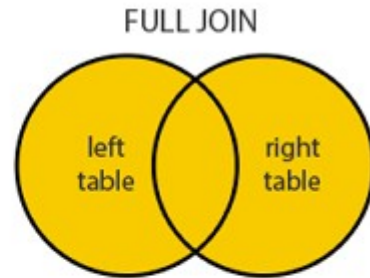
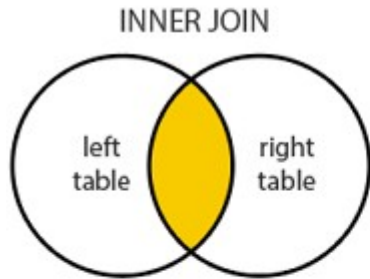


- **RIGHT OUTER JOIN** returns all records in **table2**, regardless of whether they exist in **table1**

Parent **RIGHT OUTER JOIN** *Child* returns only child records: we should use **INNER JOIN** instead.

Child **RIGHT JOIN OUTER** *Parent* returns parents regardless of whether they have child records.

Types of JOINS



- **FULL OUTER JOIN** returns all records that exist in either table

Parent **FULL OUTER JOIN** *Child* returns parents regardless of whether they have children: we should use **LEFT OUTER JOIN** instead.

Child **FULL OUTER JOIN** *Parent* returns parents regardless of whether they have children: we should use **RIGHT OUTER JOIN** instead.

Class Activity

Please work on the “Outer Join Exercise” found on the Brightspace site in the “Week 4” section on the Brightspace site.

NOTE: “Basic Select Statement” assignment is due on Friday, September 26, 2025 at 5:00 PM.

“Take-home Coding Assignment Advanced SELECT” will be available tomorrow. It is due on Friday, October 10, 2025 at 5:00 PM.

Coding Standards (as found in the course syllabus)

Only those database programming methods, practices, and techniques taught in class, or available from Brightspace notes and/or exercises, will be permissible in SDEV1201 labs and quizzes. Any solution in a lab or quiz submission using non-approved methods, practices, and/or techniques, or code that is not executable in a Postgres environment, will be given a mark of zero.